:VAL'ID *TDB-ACC-NO: NN8801262

DISCLOSURE TITLE: Method for Incorporating Timing Constraints Into Circuit Placement Objectives

PUBLICATION-DATA: IBM Technical Disclosure Bulletin, January 1988, US

VOLUME NUMBER: 30

ISSUE NUMBER: 8

PAGE NUMBER: 262 - 264

PUBLICATION-DATE: January 1, 1988 (19880101)

CROSS REFERENCE: 0018-8689-30-8-262

DISCLOSURE TEXT:

- A technique is described whereby timing constraint information is incorporated into the physical layout and chip placement of VSLI circuit networks. A procedure is applied, along with technology-generic transforms in logic synthesis, so as to reduce timing problems which can occur, due to wire length, capacitance, etc., in the physical layout of circuits. When nominal net lengths are used for timing optimization within logic syntheses, timing violations appear in the final layout of the circuit as a result of critical nets being wired with longer than nominal length. Also, many nets can exceed nominal length, if the timing of those nets is less stringent. The concept described herein provides a method for determining a maximum allowable length for each net in the design, such that if every net is wired with less than its target value, the final design will have no late signals.

Since there is a spread in the target values, such that non-critical nets are allowed greater wiring lengths, the method provides a means of allowing wiring to be longer than nominal length. By providing a length bound for each net, rather than identifying some nets as "critical", the concept avoids the potential problem of producing violations on "non-critical" nets, while reducing the length of "critical" paths. A path which has a net which exceeds its target length may still have non-negative slack because of shorter than required lengths for nets elsewhere along the circuit path. When the concept is used in conjunction with a logic transformation system, circuits or sets of circuits whose output nets have excessively large target lengths at the end of the procedure may be replaced by other circuits which occupy less area.

The target net lengths are input into a circuit placement program, so as to minimize the net length as well as minimize and balance wiring congestion. The objective is to minimize the amount by which each net exceeds its target value. The concept enables nets with less stringent timing constraints to be free to find less congested routes. The method outlined deals primarily with late-mode timing assertions, but the procedure applies equally to early-mode constraints as well. In each case, the method generates a maximum or minimum length for each net. The placement program will penalize circuit placements which require

nets to exceed or fall short of the target values. Before the procedure is applied, technology-generic transforms in logic synthesis will have been performed to solve timing problems apparent in the design of the chip.
The procedure is then applied, so as to address the values of slack which are of the size incurred when wire capacitance delay deviates significantly from that of a nominal-length network. The procedure is as follows: 1) Assign to each net a length which is the half-perimeter of the most compact placement of the circuits on the net. This will be some small multiple of the basic cell spacing. (More sophisticated values for the minimum lengths may be computed, using fanout and circuit-size information.) 2) Determine the total capacitance of each net, from the wire length of Step 1 and input gate load capacitance. 3) Compute the slack at each input of every circuit. Because all wire lengths correspond to minimum cell spacing, which is less than the nominal lengths, all slacks will be positive. 4) Flag all nets for which the slack is zero at any of the input pins to which it connects. 5) Identify the smallest slack value "S" occurring on any input pin on an unflagged net in the design. If S=0, go to Step 8.

a) Trace a backward path, toward the primary inputs, from this pin. At each circuit, follow the input net which has slack "S". Continue until a flagged net or primary input is reached. Count the number of flagged nets encountered on this backward path.

b) Trace a forward path, toward the primary outputs, from the originally identified pin. Continue until a flagged net or primary output is reached. At each circuit, in case of fanout, follow the branch of the net having slack "S". Continue until a flagged net, or primary output, is reached. Add the number of unflagged nets encountered to the number in Step 5a, resulting in a total of "N" nets. The choice of path is arbitrary in the case of ties. However, a favorable distribution of target values results if the path with the largest "N" is chosen. 6) Distribute the excess delay, positive slack "S", among the "N" nets along the path traced in Step 5. The slack can be distributed evenly by adding a capacitance to each of the nets, such that the contribution to the delay is increased by "S/N" at that net. (Proportionately more capacitance can be added to those nets having a higher fanout. Wire length is more difficult to control with simple wiring models assumed

by placement programs, which typically underestimate wire lengths.) In whatever manner the excess delay is distri buted, increasing the capacitance of these nets will reduce to zero the slack on those pins which had slack equal to "S". 7) Recalculate slacks for all input pins affected by the capacitance changes. Flag all unflagged nets which now have a slack value of zero at any input pin and return to Step 5. 8) There are no remaining unflagged nets with positive slack.

- For every net in the design, subtract the input gate load contribution from the total capacitance. The remaining capacitance corresponds to the maximum allowed wire capacitance; therefore, a maximum wire length is established for that net. 9) Use the lengths obtained in Step 8 as target lengths within the circuit placement program. If the placement is such that no net in the final design is wired with lengths exceeding its target, there will be no late signals.

:VALID *TDB-ACC-NO: NN81013900

DISCLOSURE TITLE: Method for Improving the Wirability of Chips. January 1981.

PUBLICATION-DATA: IBM Technical Disclosure Bulletin, January 1981, US

VOLUME NUMBER: 23

ISSUE NUMBER: 8

PAGE NUMBER: 3900 - 3901

PUBLICATION-DATE: January 1, 1981 (19810101)

CROSS REFERENCE: 0018-8689-23-8-3900

DISCLOSURE TEXT:

2p. Chips containing LSSD (level sensitive scan design) chained latches mus be wired without increasing the cost of test generation. This can be achieved if the latch order on the LSSD chain is allowed to be specified to optimize wiring and layout constraints. The software will allow this shuffling without lengthy recalculations of test patterns. It is well known that the performance of wiring programs can be improved by taking advantage of the input swappability of symmetric logic functions such as AND, OR. Another class of connections is described herein which can be treated as swappable, and which therefore can be used to improve the performance of these wiring programs.

- The shift register latches (SRLs) of a design that obeys the LSSD rules are connected by a logic designer in some arbitrary order to form a scan path. Test generation results in a test data file wherein the assignment of input and output values is based on that arbitrary order. Preliminary test generation normally precedes physical design because the logic designer is responsible for the testability of his logic and because of the cost of repeated passes through physical design. This invention also describes how a repetition of test generation can be avoided after an optimum order of connecting SRLs has been assigned by the wiring program. The optimum order is determined by an algorithm and is substituted for the arbitrary order specified by the logic designer.

- The simplest possible approach for evaluating the effect of the LSSD chain on placement is to include a pseudo-net during the placement procedure which includes all the LSSD input pins. The net's contribution to wirability is estimated just like the contribution of any other net. However, before the nets are serialized, or chained, each of the scan-out output nets of the LSSD SRLs receives an additional LSSD pin which may either be the scan-out pin of the chip or scan-in input of another SRL. The problem is, which input pin goes to which output pin.

- The problem of deciding the order of serialization of the LSSD latches can be formulated as a travelling salesman optimization problem. A cost can be calculated for including each of the pins in the "input set" (including the scan-out pin leaving the package) in any of the nets of the "output set" (including the scan-in pin of the

package). This cost function can be defined in a number of different ways.

- It is then possible to apply any of the numerous travelling salesman algorithms in order to obtain an approximation to the best sum over costs.
- The result of this serialization is to define the "optimum order". The test data or diagnostic information based on the optimum order is in most cases only a permutation of the bits based on the arbitrary order generated during preliminary test generation. It is then necessary to remap the bits, defining the initial and the final states of the SRLs in the test data file based on the arbitrary order to a test data file based on the optimum order. The cost of remapping is small compared to the cost of regenerating all the tests.
- The scan path can be reconnected during many iterations of placement and wiring without increasing the cost of test generation.
- The advantages of this approach are the following:

Certain constraints on the placement and wiring programs are relaxed.

The routing of the scan path can be optimized independently of other connections.

The cost of test generation is substantially reduced since most of the tests can be rearranged to fit the new scan path.

:VALID *TDB-ACC-NO: NB9306505

DISCLOSURE TITLE: LSI Low Power Oriented Layout Method with Net Switching Factors

PUBLICATION-DATA: IBM Technical Disclosure Bulletin, June 1993, US

VOLUME NUMBER: 36

ISSUE NUMBER: 6B

PAGE NUMBER: 505 - 508

PUBLICATION-DATE: June 1, 1993 (19930601)

CROSS REFERENCE: 0018-8689-36-6B-505

DISCLOSURE TEXT:

This document contains drawings, formulas, and/or symbols that will not appear on line. Request hardcopy from ITIRC for complete article.

- Disclosed is the layout method to reduce power consumption for LSI chips. The basic idea is to perform placement and wiring with the switching factor constraints of each net using Logic Simulator, which leads to shorten nets with high switching factor like CLOCK nets.

- LSI power consumption can be generally expressed by the following equation;

POWER=(1 over 2)bullet sum to N from i=1 lbrc (Ceff sub i + Cnet sub i + sum to M from j=1 Cin sub j) bullet SWf sub i rbrc bullet V sup 2 bullet F

Cnet is net capacitance
Ceff is effective capacitance, which is equivalent to power consumption in a basic macro itself.
Cin is input capacitance of basic macro.
N is the number of nets (except primary input nets).
M is the number of sink macros connected to neti.
F is the frequency of base clock in circuits.
V is the voltage that the capacitance is charged
T is the cycle time of the machine in ns

SWF means Switching Factor of each net and is defined as the number of times the circuit switches during 1000 machine cycles, divided by 1000.

TRANSITIONS means the number of transitions per 1000 machine cycles;

SWF = TRANSITIONS over 1000 (0 lt SWF le 2.00)

Switching factors of all nets in the total circuits are generated by a logic simulator and a SWF file generator. Test case will be used as input to a logic simulator to calculate switching factor.

- SWF is the number between 0 and 2.00. In LSI chip, the SWF of base clock is 2.00 because clock signals take two transitions in one cycle. SWF file generation flow is shown on Fig.1.

- This approach is focusing on high switching factor nets like CLOCK signals and first decides Target Net Capacitance so as to

shorten high switching factor nets (net switching factor is given by a logic simulator). Nets with its COST > c1 is regarded as target nets (high switching factor nets) to shorten and their target capacitances are set as the estimated capacitance value multiplied by c2 (0 < c2 < 1). Coefficients, c1 and c2 are parameters which designers define. The c1 and c2 should be optimized with characteristics of a logic network.
COST and Target Net Capacitance are defined as follows;
For each net,
COST = Capacitance(pF)intSWF
For nets with COST > c1,
TARGET = c2 int Capacitance (pF)

Placement is performed based on this Target Net Capacitance with Simulated Annealing, Min-cut method or other new placement algorithms.

- Target Net Capacitance being newly defined after placement is used in wiring phase. Wiring program tries to meet its target given in Target Net Capacitance. COST and TARGET for Target Net Capacitance are defined as same as ones at placement. Coefficients, c3 and c4 are parameters that the designer defines.
COST = Capacitance (pF)intSWF
For nets with COST > c3,
TARGET = c4 int Capacitance(pF)

Thus, this new layout method implements placement and wiring based on Target Net Capacitance considering switching factor of each net in LSI. This approach brings about around 10% power reduction of LSI.
Fig.2 shows the design flow.

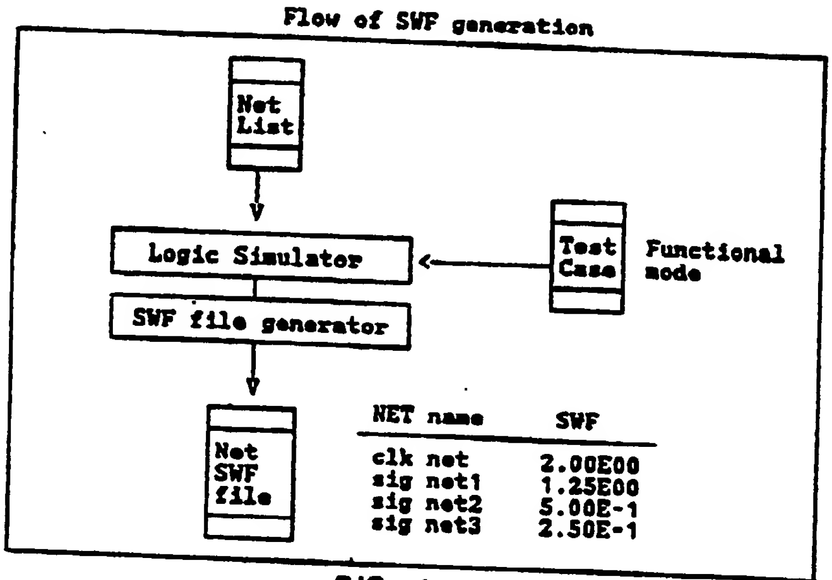


FIG. 1

$$\mathrm{POWER} = \left(\frac{1}{2}\right) \bullet \sum_{i=1}^{N} \{\, Ceff_i + Cnet_i + \sum_{j=1}^{M} Cin_j) \bullet SWf_i \} \bullet V^2 \bullet F$$

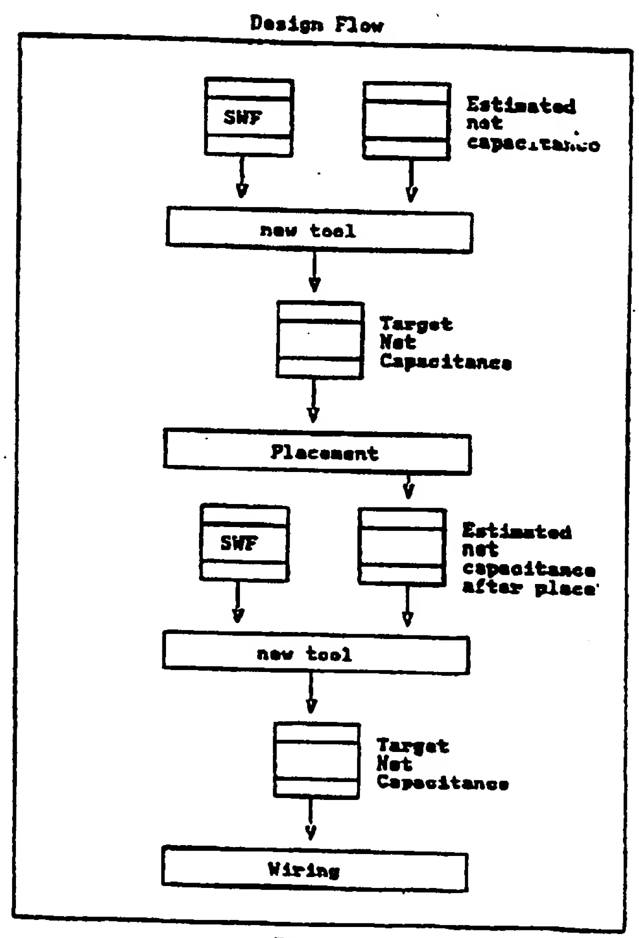


FIG. 2

:VALID *TDB-ACC-NO: NB8910106

DISCLOSURE TITLE: Floorplan-Driven Synthesis Using a Modified Form of IMPACT to Preserve Placement Information

PUBLICATION-DATA: IBM Technical Disclosure Bulletin, October 1989, US

VOLUME NUMBER: 32

ISSUE NUMBER: 5B

PAGE NUMBER: 106 - 107

PUBLICATION-DATE: October 1, 1989 (19891001)

CROSS REFERENCE: 0018-8689-32-5B-106

DISCLOSURE TEXT:

- Detailed placement is a critical but time-consuming part of the automated design methodology. This precludes it from being applied repeatedly in the design cycle. However, without physical information logic synthesis is relegated to making crude estimates about net lengths in order to get timing estimates which it can use to guide modification of logic. A way to get around this problem is to cluster the logic into relatively few groups. Then a relative placement of these clusters can be found based on their areas and connectivity, which optimizes total area, congestion and timing delays. Once such a placement of floorplan is found, synthesis can then use the delay and congestion information to modify the logic to eliminate apparent problems.

This process can be repeated until all problems are eliminated at this coarse level, after which a final detailed placement can be made.

- A key objection to this approach has been that once logic has been modified by synthesis, a new floorplan could be created that is radically different from the floorplan which guided the modifications synthesis made, thus producing new timing problems. However, IMPACT, a tool recently developed, can be modified to circumvent this problem.

- IMPACT distributes the excess delay from timing analysis on a path through the logic among the nets of this path. Currently this distribution is based on the fanout of each net and on the sensitivity of the net source to changes in load. This distributed delay is then converted into capacitance limits on each net, which are then passed to MCXA, the simulated annealing placement program. MCXA attempts to insure that these limits are met in the final placement.

- We describe a modification of IMPACT that is used to preserve information about a floorplan. The modification is that in addition to distributing delay based only on fanout and load sensitivity, the delay would also be distributed based on the bounding or steiner tree length of the nets on a path relative to the floorplan/placement from which the timing analysis was derived. Thus, everything else being equal, a net which had a longer length would get more of the excess delay than one with a shorter length, and, hence, the capacitance

limits associated with nets would in effect reflect the relative lengths of these nets in the physical placement. After timing correction, only the capacitance limits of nets connected to modified logic would be changed.

A subsequent score-driven floorplan/placement would, as part of the score, attempt to meet the generated set of capacitance limits. Since these limits are based on timing generated using a prior floorplan/placement, the new layout would reflect the prior layout and, hence, a new set of unrelated timing problems would not be created. We now explain these modifications in more detail.

- The net length would be incorporated in the delay distribution in the same manner that fanout and load is incorporated. Currently, for each net a product of its fanout and the sensitivity of the net source to load is produced and summed over all nets in the path. Delay is then distributed based on the ratio of each net's contribution to the total sum. Net length would simply be another factor in the product for each net. Thus, a greater portion of the delay would be assigned to a net which had a longer wire length in the floorplan given that fanout and load considerations were the same.

Furthermore, in order to insure that fanout and load consideration would not generate a net capacitance which would result in a significant change in net length relative to the original floorplan, lower and upper cutoffs on the capacitance assigned to a net would be produced for each net. These cutoffs would be based on the floorplan much as IMPACT presently has upper cutoffs based on load limits on the source circuit. Thus the upper and lower limits would help to drive the floorplan toward one similar to the prior floorplan.

- The changes produced by synthesis would have to be incorporated into the logic clusters before a new floorplan could be produced. Any new circuits created would be assigned to one of the logic clusters that contained the circuits from which it had been produced.

Naturally, the fewer these changes, the better the constraints based on a previous floorplan would reflect the current logic. However, due to the speed of producing a reasonable floorplan, this process could be iterated many times and thus could accommodate significant changes in early passes.

- In conclusion, this disclosure describes how floorplanning can be used to eliminate timing problems in the synthesis environment, and how a modified form of IMPACT, an IBM tool, can be used to pass placement information between successive passes of the floorplanning tool.

:VALID *TDB-ACC-NO: NA9206425

DISCLOSURE TITLE: Automatic Jog Algorithm for Layout Compactor.

PUBLICATION-DATA: IBM Technical Disclosure Bulletin, June 1992, US

VOLUME NUMBER: 35

ISSUE NUMBER: 1A

PAGE NUMBER: 425 - 432

PUBLICATION-DATE: June 1, 1992 (19920601)

CROSS REFERENCE: 0018-8689-35-1A-425

DISCLOSURE TEXT:

- Modern layout compactors are gradually being accepted by the IC designer community. Jog introduction is one of the important optimization techniques for compaction. How to efficiently generate jog has automatically gotten a lot of attention, for example (1,2,3). Force-directed jog strategy was first introduced in Computer-Aided Building-Block Artwork Generator and Editor (CABBAGE) (1). Consider the situations that a wire sits between two objects. If they fall on the critical path (or the longest path), then we can imagine that opposite forces exerted by these two objects produce a torque on the wire during the compaction. If the wire bends or jogs at some point between two force points, the critical path length can be reduced. CABBAGE failed to point out good jog points when many wires lie between two objects.

Later works (2,3) developed a "contour compaction" strategy for providing good jog candidates. The idea was borrowed from the experience of river routing. Suppose that the compaction is to the left. We push objects as far left as possible, and when we push wires to the left, they bend around "convex corners" of the contour to form river-routing patterns.

- The contour compaction approach does not work well when the wire lengths become an important factor. For a good layout, we need also to minimize the overall wire lengths in order to reduce parasitic resistance and capacitance. Less parasitic will yield better circuit performance. The capability to minimize wire lengths (4,5) is considered one of the basic ingredients in modern compactors. A good automatic jog strategy for layout compactor should take this into account. In other words, jogs need to be introduced not only to places where the overall cell size (span) can shrink, but also to places where wire lengths and parasitic can be minimized.

- First, we describe a physical model of compaction based on which a new automatic jog strategy is developed with the dual goals of minimizing both the cell span length and wire lengths. In what follows, we shall assume that the compaction direction is along the x-axis. (Analysis of y compaction essentially goes the same way.) We model the compaction process as follows:

1. A pair of enormous push forces along the x-direction are exerted on the cell bounding box to minimize the cell size.

- 2. Pulling forces along the x-direction are exerted at two ends of horizontal wires to shorten wire lengths. The sizes of these pulling forces are determined by the parasitic (resistance/capacitance) of the horizontal wires.

- 3. Vertical wires are not rigid. Under the influence of various opposing forces, net torques may be exerted on the vertical wires. Then these wires would bend (jog) to release the stress, if space permits the creation of horizontal bent wire segments. The bending (jogging) of wires changes the balance of forces and allows the layout to settle down to a better solution.

- With this physical model of compaction process in mind, the following automatic jog generation algorithm is constructed aiming at reduction in both the cell size and wire lengths. Before compaction starts, we use the scan line method to generate a set of potential jog points based on the distribution of 'forces'. Then we go ahead with the usual compaction steps: build a constraint graph with these jog points and solve the graph to minimize the critical path length and the sum of wire lengths.

- We shall employ a scan line technique to search for potential jog point candidates. Each jog point candidate is assigned a location (x, y), a direction (dir), and a pointer of the wire. In Fig. 1a, double dot lines denote a wire, and X marks the jog point. We draw a wire as double lines since wires always have finite widths. After the compaction, this jog point may transform the vertical wire into two possible jog configurations, shown in Fig. 1b and 1c, depending on the surrounding environments. For configuration Fig. 1b (c), dir is set respectively to be 1 (-1).

- Now we shall describe the way to find a set of good potential jog points (y, dir, wire-ptr). For the compaction along the x-axis, let us move the scan line from left to right. As we shall see, three kinds of events (convex corners, concave corners, and external jog points) can trigger jogs. They are collected in a linked list called the jogtriggering list. These events can create potential jog points on the neighboring vertical wires as the scan line moves along. Each convex and concave corner is given three numbers (x, y, dirc). The first two numbers tell the corner's coordinates, while the third is the direction parameter which is used for telling whether the corner is at the top (dirc = 1) or the bottom (dirc = -1) end of the mask edge.

Let the mask type of the shape for these jog-triggering events be mask1 and the mask type of the facing vertical wire be mask2, and the wire width be w.

- In the following, we shall describe how these jog-triggering events generate jog point candidates. We shall use dots and dashes to denote mask1 and mask2, respectively, and O to denote obstacles.

- 1. Convex corner. When the cell span is tightened, a sharp convex corner tends to push against the facing vertical wire producing jog point candidates as depicted in Fig. 2. Fig. 2a marks the convex corners T and B and jog points X before the compaction. Fig. 2b displays the possible layout after compaction. A top convex corner, T(u, v, dirc = 1), creates a dir = 1 jog point, while a bottom corner B(u, v, dirc = -1), creates a dir = -1 jog point and we have

dir = dirc

y = u + dirc x (rule (TB, mask2) + 0.5 x w) (1)

Here we adopt the ground rule convention in they6Ù and use mask2 to indicate the outside of mask2. So rule (mask2, mask2) is a size rule on mask2, while rule (mask1, mask2) is a space rule between two masks. If the right side of TB is an inside edge, then

TB is set to be mask1.
On the other hand, if the right side of
TB is an outside one, TB is set to be *mask1 in which case we are talking about the spacing rule between two masks in Equation (1).
- 2. Concave corner. When the horizontal wires are tightened, concave corners at their ends tend to pull the facing vertical wire with them. If jogs can be created on the vertical wire, the horizontal wire length can be shortened even more. The induced jog will have the same direction as the corner as depicted in Fig. 3a. Fig. 3b shows the effect of this jog after the compaction. The jog location is given by

dir = dirc

y = u + dirc x (rule (TB, mask2) + 0.5 x w) (2)

Here TB is the complement of TB. That is, if TB = mask, then TB = mask, and if TB = mask, then TB = mask.
- 3. External Jog Points. If the jog point occurs on the hidden segments of the wire, then we shall call it a hidden jog point. Otherwise, it is called an external jog point. When the scan line moves from one vertical wire to a second wire, an external jog point, A(xa, ya, dira), may induce new jog point, B(xb, yb, dirb), on the second wire as depicted in Fig. 4. The jog points are in the same jog direction with y location shifted to allow room for the bent wire segment. Let the width of the first wire be w1. Then we have

dirb = dira

yb = ya + dira x (rule (mask1, mask2) + 0.5 x w +
w1) (3)

Figs. 2-4 are the three ways we generate new jog points. If these newly generated jog points are not hidden, then they will be added to the jog-triggering list and used for inducing jogs when the scan line moves further.
This way, when there is a stack of vertical
wires, jog points can propagate from one vertical wire to another and produce a chain of jogs (Fig. 5).
- We have described how the jog-triggering list grows. Now we must have some means to control the growth of the list. Since only those neighboring events visible from the vertical wire can induce jogs on it, we can use the shielding mechanism to limit the growth of the jogtriggering list. In other words, after the scan line sweeps past an external edge of mask type, say maska, all those corner and jog events on maska and also falling within the interval of that edge can be safely removed from the jog-triggering list. In addition to this shielding effect, the hidden jog points will not propagate since hidden edges do not involve any design rule. Because of the use of a scan line algorithm and the shielding technique, the complexity of jog generation algorithm we described is analogous to that of constraint graph building with shadow front approach.
- In summary, we have developed a fast scan line algorithm to generate potential jog point candidates. We start with convex and concave corners as seeds and propagate through jog point chains on vertical wires. The scan line algorithm can be written as follows:
1. Map the layout into a vertical edge list.
- 2. Set the jog-triggering list to be nil.
- 3. Sort the vertical edge list and set e to be the first edge.
- 4. While e is not empty do:
a. If e is an edge coming from a vertical wire which is allowed to be jogged, then go through the jog-triggering list and generate new jog points according to Equations (1-3).
- b. If e is not a hidden edge, then remove events which

are on the same mask as e and also covered by e.

- c. If e is not a hidden edge, then add its corners plus newly generated jog points, if any, to the jog-triggering list.
- d. Go to the next edge e.
- After we use above algorithm to derive a set of potential jog point candidates, we split the vertical wires into wire segments separated by these jog points. A constraint graph is generated from ground rules with the usual shadow front approach (1,6). Each wire segment is mapped into one node of the graph. A two-step graph solver is then applied to find the optimal layout solution: first longest path algorithms (1,7) are used to find the minimum layout span, and next simplex algorithms (8) are used to find the minimum sum of wire lengths. After these two steps, those jog wire segments which can help either reduce the cell span or the sum of wire lengths will not be aligned. Horizontal bridge wires are then supplied between these jog wire segments.
- To illustrate our method, first let us study the simple example of two pairs of transistors shown in Fig.

6a. Polysilicon wires are shaded while diffusion regions are bounded by dashed lines. Curved sides of diffusion are connected to objects not drawn. According to our algorithm, the concave corner, a, will induce jog points a1, a2. Then the minimization of diffusion length will produce the layout in Fig. 6b.

- Next a CMOS circuit example before and after compaction is shown in Figs. 7a and 7b. In this layout, long and short dashes denote diffusion and polysilicon, respectively. The two rows of gates are defined by the intersections of these two layers. Metal lines are denoted by mixed long and short dashes. Contact cuts are denoted by solid lines. Parasitic weights, 1000, 100, 1 are assigned to diffusion, polysilicon, and metal layers, respectively. Because of the higher weights given to the diffusion, the upper FET gates are packed very tightly with the help of many jogs placed on the polysilicon wires. We may never find these jogs if we use the jog generation algorithm based only on the critical path analysis such as (1) or only on the contour of convex corners such as (3).
- We have implemented the automatic jog generation algorithm into a layout compactor (6) and employed it in the creation of real layout designs. Our method differs from (1) in that we take the wire lengths into account beside the critical path lengths. Our method improves over contour compaction approach (2,3) in that we also use concave corners to generate jog points. We found that it very effectively helps reduce the parasitic and enhances the performance of the circuits.
- References

(1) M. Y. Hsueh, "Symbolic Layout and Compaction of Integrated Circuits," ERL Memo, UCB/ERL M79/80, University of Berkeley, CA (December 1979).
(2) H. Shin, A. L. Sangiovanni-Vincentelli and C. H. Sequin, "TwoDimensional Compaction by Zone Refining," Proc. 23rd Design Automation Conference, 115-122 (June 1986).
(3) D. N. Deutsch, "A Compacted Channel Routing," Proc. ICCAD, 223-225 (November 1985).
(4) W. L. Schiele, "Improved Compaction by Minimized Length of Wires," Proc. 20th Design Automation Conference, 121-127 (June 1983).
(5) C. Kingsley, "A Hierarchical, Error-Tolerant Compactor," Proc. 21st Design Automation Conference, 126-132 (June 1984).
(6) J. F. Lee, "A New Framework of Design Rules for Compaction of

VLSI Layouts," IEEE Transactions on CAD 7, 11 (November 1988).
(7) J. F. Lee and D. T.
Tang, "On VLSI Layout Compaction with Grid
and Mixed Constraints," IEEE Transactions on CAD 5, 903-909
(September 1987).
(8) S. L. Lin and J. Allen, "Minplex - A Compactor that Minimizes
the Bounding Rectangle and Individual Rectangles in a Layout," Proc.
23rd Design Automation Conference, 123-128 (June 1986).

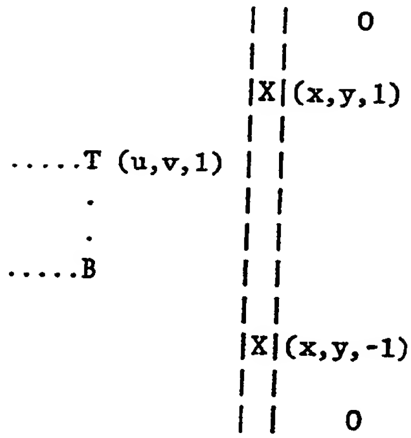
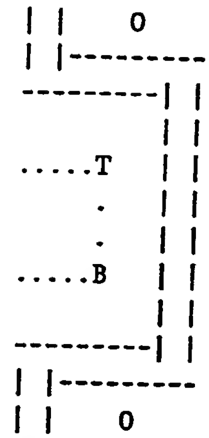


Figure 2a



2b

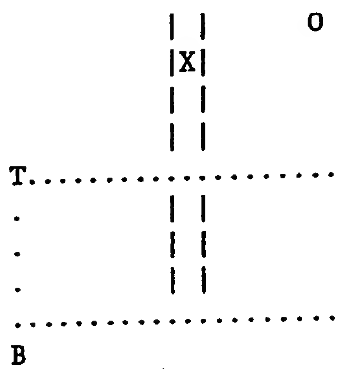
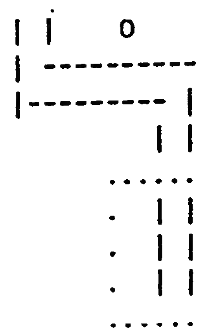


Figure 3a



3b

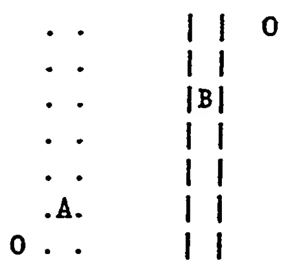
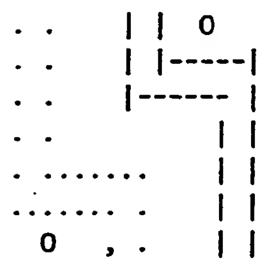
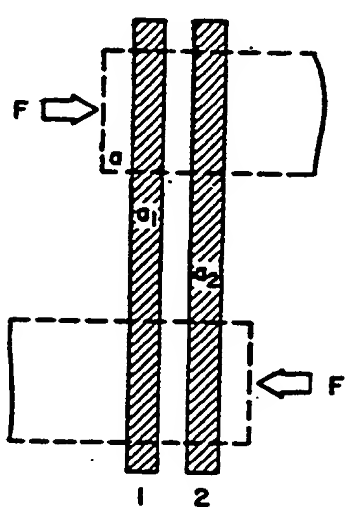


Figure 4a



4b



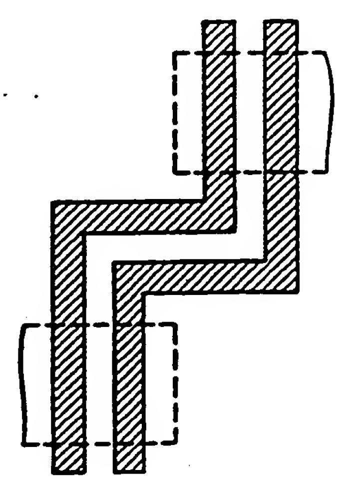


Figure 6a

6b

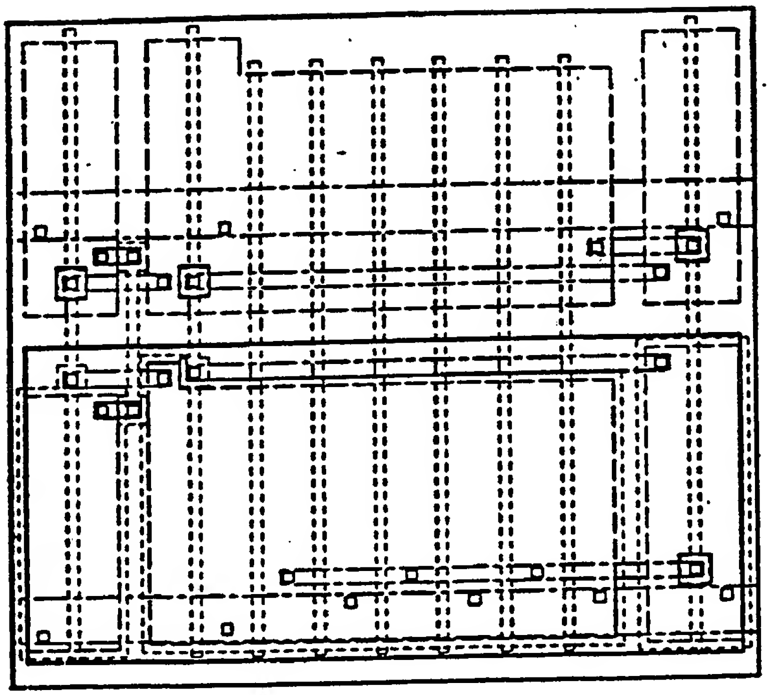


Figure 7a

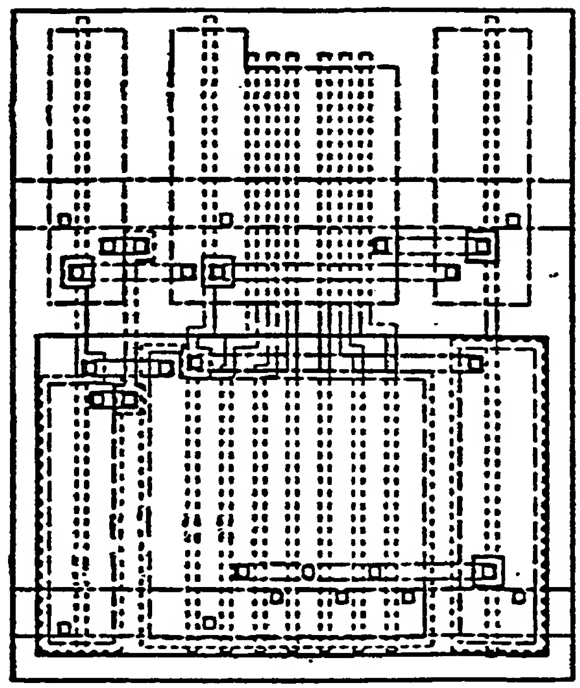


Figure 7b

:VALID *TDB-ACC-NO: NN9010221

DISCLOSURE TITLE: Floorplanning by Constraint Reduction.

PUBLICATION-DATA: IBM Technical Disclosure Bulletin, October 1990, US

VOLUME NUMBER: 33

ISSUE NUMBER: 5

PAGE NUMBER: 221 - 226

PUBLICATION-DATE: October 1, 1990 (19901001)

CROSS REFERENCE: 0018-8689-33-5-221

DISCLOSURE TEXT:

- This disclosure considers the floorplanning process that follows the relative placement of a set of blocks. A relative placement of a set of blocks is simply given by a set of horizontal and vertical topological separation constraints among the blocks. Each block is considered to have a rectangular shape. Some of these blocks may have fixed shapes and others may have flexible shapes subject to upper and lower bounds on their aspect ratios. A subset of the fixed shape blocks may even be preplaced on the chip. Define a floorplan to be legal if there are no overlaps between the blocks. The problem considered in this disclosure is that of producing a legal floorplan that respects the given topological constraint set.
- The algorithm in (*) is based on overlap resolution by the addition of constraints. The approach taken in this disclosure is based on the removal of redundant constraints.
- An interesting application of this approach is that of legalizing an early floorplan. Early floorplanning is a technique that is becoming increasingly important for timing critical designs. The idea here is to floorplan a set of functional blocks even before the synthesis of the logic inside these blocks. Such an early floorplan gives the logic designer an idea of how the physical design process can impact the timing on certain critical paths of the logic. This information can be used to drive logic synthesis of these blocks, so as to optimize the logic on these critical paths. The early floorplanning process has to use rough estimates for the area of the blocks since they have not even been synthesized yet. After synthesis the areas of certain blocks may increase and those of others may actually reduce.

Thus, the early floorplan is no longer legal because it now contains overlaps due to expansion of some of the blocks. The approach described in this disclosure can be applied by deriving a constraint set from the early floorplan and using this to legalize the floorplan.
- The floorplanning problem is formulated as follows: ***** SEE ORIGINAL FOR MATHEMATICAL EQUATIONS IN DOCUMENT *****

Given a set of blocks B, let u(bi) and l(bi) be upper and lower bounds on the aspect ratio of a block bi . If u(bi) = l(bi), then the block bi is said to be a fixed shape block, else it is a flexible shape block. A subset Bp of the fixed shape blocks are

required to be pre- placed, i.e., their coordinates on the plane are fixed and cannot be changed.

Topological Constraint Set: A topological constraint set of a set of blocks is given by two directed acyclic graphs (DAGs) (GH,GV): GH is the horizontal constraint graph and GV is the vertical constraint graph. The node set of both GH and GV are exactly the set of blocks B. If (bi,bj) is an edge in GH, then bi is to be placed to the left of bj .

If (bi,bj) is an edge in GV, then bi is to be placed below bj .

Transitive Closures: Let (GH)+ and (GV)+ be the transitive closures of GH and GV, respectively. In other words (bi,bj) is an edge in (GH)+ if and only if there is a directed path from bi to bj in GH . (GV)+ is similarly defined.

- Completeness: A constraint set (GH,GV) is said to be complete if for any pair of blocks bi,bj there is an edge (i.e., constraint) involving them in either (GH)+ or (GV)+ or both. In other words, (GH)+ or (GV)+ is a complete DAG.

Strong completeness: A constraint set (GH,GV) is said to be strongly complete if for any pair of blocks bi,bj there is an edge (i.e., constraint) involving them in either GH or GV or both. In other words, GH GV is a complete DAG.

The notion of completeness allows separations between blocks to be achieved through transitivity of constraints, while the notion of strong completeness requires the explicit presence of a constraint for each pair of blocks. Clearly, if a constraint set is strongly complete, it is also complete, but not vice versa.

These two notions lead to different versions of the problem, and consequently lead to different notions of constraint redundancies and different versions of the algorithm.

- Legal Floorplan: A floorplan is defined to be legal if it has no overlaps, the block dimensions satisfy the given bounds on aspect ratios, and preplaced macros are kept at their required coordinates.

Respect: A legal floorplan of B is said to respect a complete constraint set (GH,GV), if for any pair of blocks bi,bj, they are separated in the floorplan either according to their constraint in (GH)+ or their constraint in (GV)+ .

Strong Respect: A legal floorplan of B is said to strongly respect a complete constraint set (GH,GV), if for any pair of blocks bi,bj, they are separated in the floorplan either according to their constraint in GH or their constraint in GV .

Note that if a pair of blocks are constrained in both the horizontal and vertical direction, the above definition does not require both constraints to be met in a respecting floorplan.

- Given a constraint set, a floorplan that respects it can be generated by topological sort of the DAGs GH and GV . Topological sort of a DAG with n vertices and m edges can be performed in O(n + m) time. If the constraint set is (strongly) complete, then the floorplan will have no overlaps and thus be legal. The goal is to find a respecting floorplan that has minimum area.

Statement of the Problem: Given a (strongly) complete constraint set of a set of blocks, find a legal floorplan of minimum area that (strongly) respects the constraint set. In practice, it may be the case that the chip dimensions are given, in which case the problem is to find a legal respecting floorplan that fits within the given dimensions.

The key idea used in this approach is that of removing redundant constraints from a given constraint set.

Constraints are nothing but
edges of the DAGs GH and GV .
Two notions of constraint redundancy that correspond to the notions of strong completeness and completeness are now defined.
Strong Redundancy: An edge e in GH is said to be strongly redundant if GH - e GV = GH GV . Strong redundancy of an edge in GV is similarly defined.
- In other words, an edge is strongly redundant if it is present in both GH and GV . A strongly complete constraint set if (GH,GV) will remain strongly complete after the removal of a strongly redundant edge. It is also obvious that a floorplan that strongly respects a constraint set (GH,GV) minus a strongly redundant edge will strongly respect (GH,GV) minus a strongly redundant edge will strongly respect (GH,GV) itself. Therefore, strongly redundant edges can be removed from a strongly complete constraint set and yet always yield a floorplan that strongly respects the original constraint set.
Transitive Redundancy: An edge e in GH is said to be transitive redundant if (GH - e+ (GH)+ = (GH)+ (GV)+ . Transitive redundancy of an edge in GV is similarly defined.
- In other words, an edge is transitive redundant if its removal does not affect the union of the transitive closures. A complete constraint set if (GH,GV) will remain complete after the removal of a transitive redundant edge. It is also obvious that a floorplan that respects a constraint set (GH,GV) minus a transitive redundant edge will respect (GH,GV) itself. Therefore, transitive redundant edges can be removed from a complete constraint set and yet always yield a floorplan that respects the original constraint set.
Note that an edge which is strongly redundant need not be transitive redundant, and vice versa.
The notion of transitive redundancy can be used for constraint removal from both complete and strongly complete constraint sets.
The notion of strong redundancy can be used only on strongly complete constraint sets.
However, an edge can be checked for strong redundancy in constant time. On the other hand, checking for transitive redundancy requires the computation of transitive closures. In practice, the input constraint sets are invariably strongly complete. In other words, some relation can always be extracted for each pair of blocks from any reasonable input description (e.g., an illegal floorplan).
- The goal of the floorplanning algorithm is to optimize floorplan area. Therefore, most of the action in the algorithm takes place on paths in GH and GV that are critical to area reduction. The length of a path of blocks in GH or GV is defined to be the sum of the dimensions of the blocks and the separations between the blocks. The longest path in GH or GV is called a critical path. The critical path in a DAG can be computed using a topological sort algorithm. The two main steps in their order of application in the floorplanning algorithm are:
1. Constraint reduction on critical paths.
2. Aspect ratio adjustment of blocks on critical paths.

An algorithm that uses the notions of strong completeness and strong redundancy is outlined below. A similar algorithm can be derived for the nations of completeness and transitive redundancy.
- Algorithm Floorplan_Reduce_Strong:
Input: A strongly complete constraint set (GH,GV) of a set of blocks B.
Repeat steps 1 and 2 until there is no improvement in the floorplan area.

1. Repeat following steps a,b,c
a. Topological sort GH,GV . Let PH,PV be critical paths of GH,GV .
b. Select PH or PV whichever is critical.
c. Remove one strongly redundant edge on the selected critical path until no strongly redundant edges on PH, or PV exist.
2. for j: = 1 to number_of_aspect_ratio_iterations do
a. Store the current aspect ratios.
b. Let PH, PV be critical paths of GH,GV .
c. Select PH or PV whichever is more critical.
d. Adjust aspect ratios of flexible blocks on selected path.
e. Topological sort GH and GV .
f. If new floorplan is the best seen so far, update stored aspect ratios.

- Step 1 of the algorithm does the constraint reduction, and step 2 does the aspect ratio adjustments. A pass of the algorithm constitutes one execution of the two steps. Passes are repeated until there is no improvement in the floorplan area. Typically, only 3 or 4 passes are required. Aspect Ratio Adjustment: The aspect ratios of each flexible block on the selected critical path are perturbed by a given constant factor, so as to reduce the length of the path. The dimensions in the direction of the selected path are thus reduced, and those orthogonal to the directions of the selected path are increased. During the iterations of aspect ratio adjustments, the best set of aspect ratios is stored and is used in the next pass of the algorithm. Note that the constraint set remains unchanged during these iterations.
- Construction of input constraint set: The input to the above algorithm is a strongly complete constraint set. In practice, the input comes from the output of a relative placement program that produces a floorplan with overlaps. The following simple algorithm is used to convert an illegal floorplan into a constraint set. Each pair of blocks is examined. If the blocks are separated, then topological constraints are added to GH or GV or both accordingly. If the two blocks overlap, then topological constraints are added according to the relative positions of their centers. It is easy to see that there is either a horizontal or a vertical constraint for each pair, thus implying that the result is a strongly complete constraint set.
- Reference

(*) S. K. Dong, J. Cong, C. L. Liu, "Constrained Floorplan Design for Flexible Blocks," Digest of Technical Papers, ICCAD, 488-491 (1989).

:VALID *TDB-ACC-NO: NN9201221

DISCLOSURE TITLE: Optimization Algorithm for Circuit Transistor Sizes.

PUBLICATION-DATA: IBM Technical Disclosure Bulletin, January 1992, US

VOLUME NUMBER: 34

ISSUE NUMBER: 8

PAGE NUMBER: 221 - 224

PUBLICATION-DATE: January 1, 1992 (19920101)

CROSS REFERENCE: 0018-8689-34-8-221

DISCLOSURE TEXT:

- Disclosed is an algorithm for obtaining the optimal solution on an approximate surface (see preceding article) to solve the transistor size optimization problem for a basic cell. The optimal solution on the approximate surface will be given in closed form.
- First we define the time delay function td as: ***** SEE ORIGINAL FOR MATHEMATICAL EQUATIONS IN DOCUMENT ***** where wi, i = 1,...,n, is the width of the ith type transistor of a cell.
- The following time delay function was selected due to its simplicity and well-defined mathematical properties (see preceding article).
- The optimization problem to be performed is to minimize the sum of the widths of all transistors in a cell while meeting the time delay constraint , or where mi is the number of transistors of type i having width wi, and n is the number of transistor types in a cell.
- There are several standard methods of solving the above optimization problem (1,2). The Lagrange multiplier method is chosen for its simplicity. To this end we formulate the Lagrange function m such that where g is the Lagrange multiplier constant. The necessary condition for an optimal is since the time delay function f must satisfy

As one can see from Eq(5) and Eq(6) that we have n+1 equations with n+1 unknowns. Consequently, there should exist a unique solution.

- Solving Eq(5) and Eq(6) for wi, we can obtain the closed form solution for the above optimal problem. where

All transistor widths wi, i=l,...., n can be computed from Eq(7) if the Lagrange multiplier g is known. A set of optimized solutions with their time delay constraints can be obtained from the above equations without running further circuit analysis. Consequently, we can compute a set of optimal points on the approximate surface for different values of .

- We optimize our problems on an approximation surface. In theory, we can obtain all the optimal transistor sizes, for a new

design point, in only one iteration. However, due to the relatively large convergence error of the circuit analysis program, there exists numerical noise in the computation of the partial derivatives calculated from small numerical values in the output data of the circuit evaluation. This noise may cause computation problems during the process of optimization. Furthermore, the optimal process is also affected by the upper and lower bounds of the transistor size specified by the designer. The folloing algorithm was implemented in an experimental program:

ALGORITHM OPTIMIZATION

Step 1.

- Initialization

  Noise_Flag (i) = O, i=i,....,n;

  Sum1 = O;

  Sum2 = O;

Step 2.

- Test values of partial derivatives for possible noise.

  Do for i = 1,.....,n;

  then set the Noise_Flag (i) - 1:

where e is a small real number

Step 3.

- Compute Sum 1, and Sum 2.
- Set i = 1;

  While i < = n do

  Begin

  if Noise_Flag (i) = 0 then

Step 4.

- Compute the Lagrange multiplier g from Eq(8)

Step 5.

- Compute transistor widths wi, i = 1,...,n.from Eq(7)

  Set i=1;

  While i < = n

  Do Begin

  If Noise_Flag = 0 then

References

(1) W. T. Nye, D. C. Riley, A. Sangiovani-Vincentelli and L. Tits, "DELIGHT.SPICE: An Optimization-Based System for the Design of Integrated Circuits," IEEE Trans . CAD 7, 4, 501-519 (April 1988).

(2) G. D. Hachtel, T. R. Scott and R. P. Zug, "An Interactive Linear Approach to Model Parameter Fitting and Worst Case Circuit Design," IEEE Trans . on Circuits and Systems CAS-27, 10, 871-881 (October 1980).

$$f(w_1, w_2, \ldots, w_n) = f^0(w_1^0, w_2^0, \ldots\ldots, w_n^0) + \sum_{i=1}^{n} \frac{\partial f}{\partial w_i}(w_i - w_i^0) + 1/(2!) \sum_{i=1}^{n} \frac{\partial^2 f}{\partial w_i^2}(w_i - w_i^0)^2 \qquad (2)$$

$$\min \sum_{i=1}^{n} m_i w_i \qquad (3)$$
$$s.t. \quad f(w_1, \ldots, w_n) = t_d^{\times}$$

$$\mu = \sum_{i=1}^{n} m_i w_i + \lambda f(w_1, \ldots\ldots, w_n) \qquad (4)$$

$$w_i = w_i^0 - \frac{\partial f}{\partial w_i}\Big|_{w_i = w_i^0} \left(\frac{\partial^2 f}{\partial w_i^2}\right)^{-1}\Big|_{w_i = w_i^0} - m_i \lambda^{-1} \left(\frac{\partial^2 f}{\partial w_i^2}\right)^{-1}\Big|_{w_i = w_i^0} \qquad (7)$$

$$\lambda^2 = \frac{-1/2 \sum_{i=1}^{n} m_i^2 \left(\frac{\partial^2 f}{\partial w_i^2}\right)^{-1}}{f^0 - t_d^{\times} - 1/2 \sum_{i=1}^{n} \left(\frac{\partial f}{\partial w_i}\right)^2 \left(\frac{\partial^2 f}{\partial w_i^2}\right)^{-1}} \qquad (8)$$